Places: Parallelism for Dynamic Languages

Kevin Tew and Matthew Flatt

THE UNIVERSITY OF UTAH

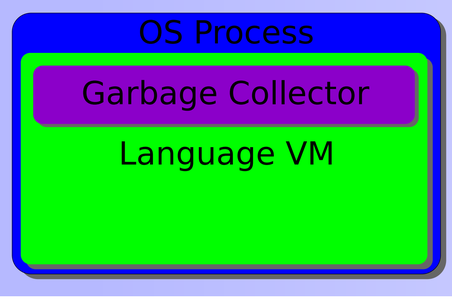
Programmers need a low cost means to prototype and experiment with parallel programs. Places add message passing style parallelism to dynamic languages such as PLT Scheme. In places, tasks have process like isolation but are implemented as operating system threads.

Programming with locks is an endless task.

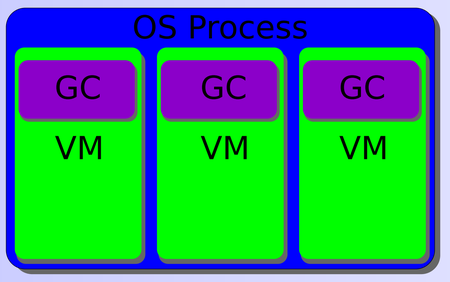
Instead of adding locks, Places migrate dynamic language runtimes to a process isolation model of parallelism.

Add Global Collector

Existing Language VM

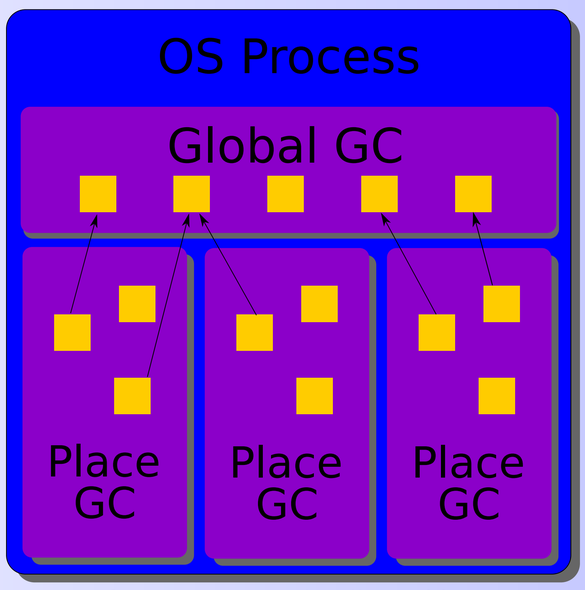


Replicate VM in Process

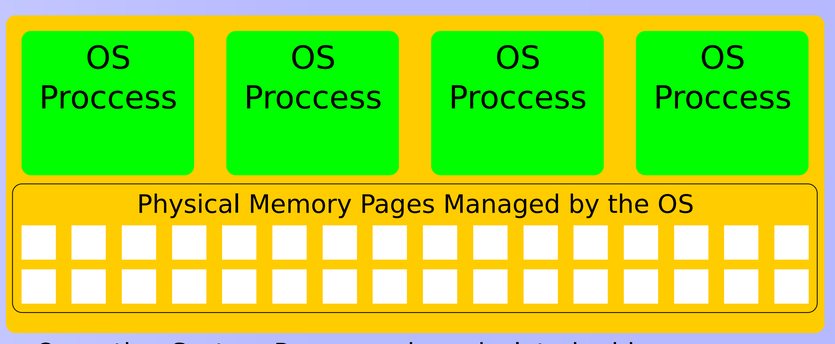


Exploiting Sharing Under the Hood

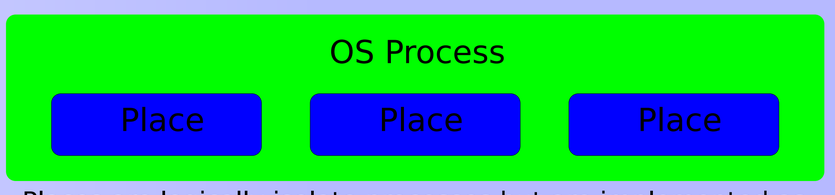
PLT Places



References are only allowed in one direction, from places to the global GC



Operating System Processes have isolated address spaces. OSs, conserve memory by mapping identical data from different processes to the same underlying physical page.



Places are logically isolate processes but are implemented as OS threads in a shared memory space.

Jitted Code Blocks

```
lea     -0x120(%rbp),%rdx
mov     %rcx,%rsi
mov     %rax,%rdi
callq   0x49202a
cmpl    $0x0,-0x210(%rbp)
jne     0x492a28
mov     -0x18(%rbp),%rax
movb    $0x1,0x4(%rax)
mov     0x4b3311(%rip),%rax
test    %rax,%rax
je      0x492a43
mov     0x4b3305(%rip),%rax
callq   *%rax
mov     %rax,-0x30(%rbp)
jmp     0x492a4b
movq    $0x0,-0x30(%rbp)
mov     0x48658e(%rip),%rax
```

Global Symbol Table

'lambda
'define
'struct
'violet
'loop
'dv-revb
'utah
'ccssgbe

VM Bytecode

```
 35 set I0, P2
 38 set I2, P1
 41 le I2, I0, 15
 45 set I3, P0[I0]
 49 add I1, I3
 52 inc I0
 54 branch -13
 41 le I2, I0, 15
 45 set I3, P0[I0]
 49 add I1, I3
 52 inc I0
 54 branch -13
 41 le I2, I0, 15
 56 set_returns PC12 (1)
 59 returncc
186 set_returns PC4 (1)
```

Jitted code blocks, global symbol tables and vm bytecode are a few examples of language runtime data that can be shared across places.

Places API

```
(place module-path start-proc place-channel)
Starts running start-proc in parallel. Start-proc must be a
function defined in module-path.  The place procedure
returns immediately with a place descriptor value.

(place-wait p)
Returns the return value of a completed place p, blocking
until the place completes (if it has not already completed).

(place? x)
Returns #t if x is a place object.

(place-channel place2)
Creates a channel between the current place and place2

(place-channel-send ch x)
Sends an immutable message x on channel ch.

(place-channel-recv ch)
Returns an immutable message received on channel ch.

(place-channel? x)
Returns #t if x is a place-channel object.
```

Example

```
(define (places-prime n)
  (let ([pls (for/list ([i (in-range 2 (- n 1))])
     (let* ([p (place
                 "prime-worker.ss"
                 'prime-main)])
       (place-channel-send p (list i n))))])
         (andmap place-channel-recv pls))))


(module prime-worker
  (define (prime-main ch)
    (match (place-channel-recv ch)
      [(list i n)
         (place-channel-send ch
           (not (= 0 (remainder n i))))])))


(places-prime 5)
```

"Places: As fast as threads, as reliable as processes."